Pseudo basic steps: Bound improvement guarantees from Lagrangian decomposition in convex disjunctive programming

Dimitri J. Papageorgiou and Francisco Trespalacios

Corporate Strategic Research

ExxonMobil Research and Engineering Company

1545 Route 22 East, Annandale, NJ 08801 USA

{dimitri.j.papageorgiou,francisco.trespalacios}@exxonmobil.com

June 15, 2017

Abstract

An elementary, but fundamental, operation in disjunctive programming is a basic step, which is the intersection of two disjunctions to form a new disjunction. Basic steps bring a disjunctive set in regular form closer to its disjunctive normal form and, in turn, produce relaxations that are at least as tight. An open question is: What are guaranteed bounds on the improvement from a basic step? In this paper, using properties of a convex disjunctive program's hull reformulation and multipliers from Lagrangian decomposition, we introduce an operation called a *pseudo* basic step and use it to provide provable bounds on this improvement along with techniques to exploit this information when solving a disjunctive program as a convex MINLP. Numerical examples illustrate the practical benefits of these bounds. In particular, on a set of K-means clustering instances, we make significant bound improvements relative to state-of-the-art commercial mixed-integer programming solvers.

Keywords: basic step, disjunctive programming, K-means clustering, Lagrangian decomposition, mixed-integer conic quadratic optimization.

1 Introduction

Disjunctive programming is optimization over disjunctive sets, i.e., sets of inequalities connected to one another by the operations of conjunction or disjunction. Conceived by Balas in the early 1970s, disjunctive programming was originally advertised as "a class of problems which subsumes pure and mixed-integer programs and many other nonconvex programming problems" [4]. Within the integer programming community, Balas's seminal ideas have gone on to influence a number of topics ranging from lift-and-project cuts [6] and disjunctive cuts to intersection cuts for convex lattice-free sets and more. For an enjoyable first-hand account of disjunctive programming, the reader is encouraged to peruse [5, Chapter 10].

In the process systems engineering community, which has also made numerous contributions to the field of optimization, another branch of disjunctive programming was formed that goes by the name *generalized* *disjunctive programming* (GDP). Raman and Grossmann [17] generalized Balas's initial paradigm in two important ways. First, whereas Balas focused on a purely linear setting, often referring to disjunctive programming as "optimization over a union of polyhedra" or "linear programming with disjunctive constraints," Raman and Grossmann extended this framework to include arbitrary nonconvex sets. Second, and perhaps more importantly, they introduced logic propositions in terms of Boolean variables to provide "an alternative modeling framework to mixed-integer linear/nonlinear programming that is more adept at translating physical intuition of engineering principles into rigorous mathematical formalism" [20]. To be precise, a student of mixed-integer programming (MIP) is taught to write down a model expressed solely in terms of two primitives: continuous/integer decision variables, which are either real- or integer-valued, and algebraic constraints. Meanwhile, a student of GDP is taught to model with, in addition to the aforementioned primitives, disjunctions (including implications) and Boolean variables. With these additions, GDP is arguably a more expressive modeling paradigm than mixed-integer programming. For a more detailed discussion on GDP modeling, see Grossmann and Trespalacios [13]. In this paper, we do not consider Boolean variables and logic propositions as found in GDP; hence, we omit the term "generalized."

There are several other noteworthy papers on or involving convex disjunctive programming. Ceria and Soares [11] showed how to construct the convex hull of the union of a finite number of convex sets by exploiting the perspective function. Stubbs and Mehrotra [21] devised a branch-and-cut algorithm for 0-1 mixed-integer convex optimization problems. Ruiz and Grossmann [18] extended hierarchies of disjunctive programming and linear GDP in [3] and [20], respectively, to nonlinear convex GDPs. Trespalacios and Grossmann [22] proposed an algorithmic approach to improve convex GDP formulations that involves, among other steps, the iterative application of basic steps and the judicious selection of where to apply them. Our work can be viewed as an extension of the Lagrangian relaxation introduced by Trespalacios and Grossmann [23] for *linear* GDPs. Bonami et al. [10] and Belotti et al. [7] present techniques for handling indicator constraints, i.e., constraints that either hold or are relaxed depending on the value of a binary variable. Since an implication is logically equivalent to a disjunction, many of the techniques suggested make use of disjunctive programming concepts. The findings in this paper are meant, in part, to support the claim made in Belotti et al. [7] "... that aggressive bound tightening is often overlooked in MIP, while it represents a significant building block for enhancing MIP technology when indicator constraints and disjunctive terms are present."

Consider a nonlinear disjunctive program

$$\begin{aligned} \min \quad & f(\mathbf{x}) && \text{(1a)} \\ \text{s.t.} \quad & g(\mathbf{x}) \le \mathbf{0} && \text{(1b)} \\ & \mathbf{x} \in \mathcal{F} = \bigcap_{k \in \mathcal{K}} \bigcup_{i \in \mathcal{D}_k} \mathcal{S}_{ki} \ , && \text{(1c)} \end{aligned}$$

where f is a nonlinear function, $g(\mathbf{x}) \le \mathbf{0}$ denotes a system of "global" constraints that are independent of the disjunctions, $\mathcal{K} = \{1, \dots, K\}$ is a set of disjunctions, $\mathcal{D}_k$ is the set of disjuncts associated with the kth disjunction, and $\mathcal{S}_{ki} \subset \mathbb{R}^n$ is a closed and bounded set describing the feasible region of the ith disjunct in the kth disjunction. In this paper, we focus on the special case when f is a convex function and all disjuncts are convex, which can be written as:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} && \text{(2a)} \\ \text{s.t.} \quad & \mathbf{x} \in \bigcap_{k \in \mathcal{K}} \bigcup_{i \in \mathcal{D}_k} \mathcal{C}_{ki} \ . && \text{(2b)} \end{aligned}$$

Notice that in (2), we consider a linear objective function since one can always minimize an auxiliary variable z and include a constraint $z \geq f(\mathbf{x})$. Likewise, there are no global constraints $g(\mathbf{x}) \leq \mathbf{0}$. This is without loss of generality since global constraints can always be handled in at least two alternate ways: they can be (1) treated as a single (improper) disjunction possessing a single disjunct, or (2) embedded within each disjunct, although this obviously comes at the expense of working with disjuncts having more inequalities.

To obtain a disjunctive program's so-called *hull reformulation*, it is convenient, but not necessary, to work with an extended formulation in which "copies" (also known as *disaggregated variables*) of the original decision variables are created, one copy for each disjunct. This leads to the following formulation:

$$\begin{aligned} z^* = \min \quad & \mathbf{c}^\top \mathbf{x} && && \text{(3a)} \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{v}_k && \forall\, k \in \mathcal{K} && \text{(3b)} \\ & \mathbf{v}_k \in \mathcal{F}_k = \bigcup_{i \in \mathcal{D}_k} \mathcal{C}_{ki} && \forall\, k \in \mathcal{K} && \text{(3c)} \end{aligned}$$

The hull relaxation can then be expressed as

$$\begin{aligned} (HR_{\mathcal{K}}) \quad z_{\mathcal{K}}^{\text{Hrel}} = \min \quad & \mathbf{c}^\top \mathbf{x} && && \text{(4a)} \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{v}_k && \forall\, k \in \mathcal{K} \quad (\boldsymbol{\lambda}_k \in \mathbb{R}^n) && \text{(4b)} \\ & \mathbf{v}_k \in \operatorname{clconv}(\mathcal{F}_k) && \forall\, k \in \mathcal{K}\ , && \text{(4c)} \end{aligned}$$

where clconv denotes the convex closure of a set. Lagrange multipliers, if they exist, that link the original variables $\mathbf{x}$ with their copies $\mathbf{v}_k$ are expressed as $\boldsymbol{\lambda}_k \in \mathbb{R}^n$.

Remark 1. Solving the hull relaxation (4) is a convex optimization problem and, thus, can be done in polynomial time.

Our results assume that the original convex disjunctive program (2) has been reformulated so that we can exploit the multipliers $\boldsymbol{\lambda}_k$ corresponding to the constraints (4b). In addition, the prevailing theory typically assumes an extended formulation using disaggregated variables to obtain a tractable algebraic representation of the hull reformulation [11, 18, 21].

Given the progress in general purpose optimization solvers, disjunctive programs are typically re-formulated as mixed-integer nonlinear programs (MINLPs) when solved numerically. To achieve this, each disjunct is encoded with an algebraic description of the form $\mathcal{C}_{ki} = \{\mathbf{x} \in \mathbb{R}^n : g_{ki}(\mathbf{x}) \leq \mathbf{0}\}$, where g_{ki} are convex functions for all $k \in \mathcal{K}$ and $i \in \mathcal{D}_k$. As shown in Ruiz and Grossmann [18], based on the work of [11] and [21], a convex disjunctive program can be expressed as the following convex MINLP:

$$\begin{aligned} \min_{\mathbf{v},\mathbf{x},\mathbf{y},\boldsymbol{\nu}} \quad & \mathbf{c}^\top \mathbf{x} && && \text{(5a)} \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{v}_k && \forall\, k \in \mathcal{K} \quad (\boldsymbol{\lambda}_k \in \mathbb{R}^n) && \text{(5b)} \\ & \mathbf{v}_k = \sum_{i \in \mathcal{D}_k} \boldsymbol{\nu}_{ki} && \forall\, k \in \mathcal{K} && \text{(5c)} \\ & (\text{cl } g'_{ki})(\boldsymbol{\nu}_{ki}, y_{ki}) && \forall\, k \in \mathcal{K}, i \in \mathcal{D}_k && \text{(5d)} \\ & \sum_{i \in \mathcal{D}_k} y_{ki} = 1 && \forall\, k \in \mathcal{K} && \text{(5e)} \\ & -\mathbf{L} y_{ki} \leq \boldsymbol{\nu}_{ki} \leq \mathbf{L} y_{ki} && \forall\, k \in \mathcal{K}, i \in \mathcal{D}_k && \text{(5f)} \\ & y_{ki} \in \{0,1\} && \forall\, k \in \mathcal{K}, i \in \mathcal{D}_k\ , && \text{(5g)} \end{aligned}$$

where generically (cl g')$(\mathbf{x}, y)$ denotes the closure of the perspective function g' of $g(\mathbf{x})$ at $(\mathbf{x}, y)$ and is given by $g'(\mathbf{x}, y) = yg(\mathbf{x}/y)$. In particular, (cl g'_{ki})$(\boldsymbol{\nu}_{ki}, y_{ki}) = y_{ki} g_{ki}(\boldsymbol{\nu}_{ki}/y_{ki})$ if $y_{ki} > 0$ and (cl g'_{ki})$(\boldsymbol{\nu}_{ki}, y_{ki}) = 0$ if $y_{ki} = 0$. Here, $\mathbf{L}$ denotes a vector of finite bounds on the absolute value of $\mathbf{x}$, which exists due to the assumption that all disjuncts are compact.

For a general convex disjunctive program, attempting to solve hull formulation (5) can lead to numerical difficulties when employing a branch-and-cut algorithm for at least two reasons. First, the resulting convex MINLP is large leading to large, possibly ill-conditioned, convex subproblems at each search tree node. Second, the functions (cl g'_{ki})$(\boldsymbol{\nu}_{ki}, y_{ki})$ are not differentiable at $y_{ik} = 0$. A popular approach to combat these non-differentiabilities was introduced in Sawaya [19] and relies on approximating the functions (cl g'_{ki}) in such a way that the formulation is tight when $y_{ik} \in \{0, 1\}$, but slightly relaxed at non-binary solutions.

When working with specially structured convex disjunctive programs, many of these closure and differentiability concerns disappear. For linear disjunctive programs (i.e., all disjuncts $\mathcal{C}_{ki}$ are polytopes), Balas [2] provides an explicit extended formulation for the hull reformulation. For conic quadratic disjunctive programs, each disjunct is the intersection of one or more conic quadratic constraints $\mathbf{x}^\top \mathbf{Q} \mathbf{x} + \mathbf{b}^\top \mathbf{x} + \gamma \leq 0$. Omitting subscripts, consider a disjunct with a single conic quadratic constraint. The set corresponding to the perspective function of a conic quadratic function is given by $\left\{(\mathbf{x}, y) \in \mathbb{R}^n \times [0, 1] : \mathbf{x}^\top \mathbf{Q} \mathbf{x}/y + \mathbf{b}^\top \mathbf{x} + \gamma y \leq 0\right\}$. Although this set is not closed at $y = 0$ since $\mathbf{x}^\top \mathbf{Q} \mathbf{x}/y$ is not defined at $y = 0$, it is possible to lift this set into a higher dimensional space and eliminate this issue: $\left\{(\mathbf{x}, y, t) \in \mathbb{R}^n \times [0, 1] \times \mathbb{R}_+ : t + \mathbf{b}^\top \mathbf{x} + \gamma y \leq 0, \mathbf{x}^\top \mathbf{Q} \mathbf{x} \leq ty\right\}$. This set is conic quadratic representable as the constraint $\mathbf{x}^\top \mathbf{Q} \mathbf{x} \leq ty$ is a rotated second-order cone constraint (Ben-Tal and Nemirovski [8]). We will use this result multiple times in our computational experiments. Finally, we have deliberately chosen to work with the consolidated formulation (3), in which constraints (3c) subsume constraints (5c)-(5g), as the bulk of our results do not require the latter representation.

2 Basic Steps

Basics steps are not a commonly used tool in a mixed-integer programmer's toolbox. A quick look in any of the major solvers' API reveals that a basic step is not even a callable operation. Thus, it is natural to ask: Why should one care about basic steps in the first place? We attempt to provide some answers to this question and revive interest in this basic operation.

As described by Balas [3], a disjunctive set $\mathcal{F}$ can be expressed in many different forms that are logically equivalent and can be obtained from each other by considering $\mathcal{F}$ as a logical expression whose statement forms are inequalities, and applying the rules of propositional calculus. Among these equivalent forms, the two extremes are the *conjunctive normal form* (CNF)

$$\mathcal{F} = \bigcap_{k \in \mathcal{K}} \mathcal{E}_k \ ,$$

where each $\mathcal{E}_k$ is an elementary disjunctive set, which, for convex disjunctive programming, is the union of finitely many convex inequalities $g_j(\mathbf{x}) \leq 0$, and the *disjunctive normal form* (DNF)

$$\mathcal{F} = \bigcup_{i \in \mathcal{D}} \mathcal{C}_i \ ,$$

where each $\mathcal{C}_i$ is a convex set (and compact, in our setting). Why is DNF important? Balas [3, Theorems 3.3 and 3.4] showed that the convex hull of a linear disjunctive program in DNF is the projection of a higher dimensional polyhedron onto $\mathbb{R}^n$. Practically speaking, this means that solving the hull relaxation of a linear disjunctive program in DNF, which is a linear program (albeit a potentially very large one), yields an optimal solution to the original problem. Meanwhile, most discrete optimization problems are stated in the form of an intersection of elementary disjunctions, that is, in CNF.

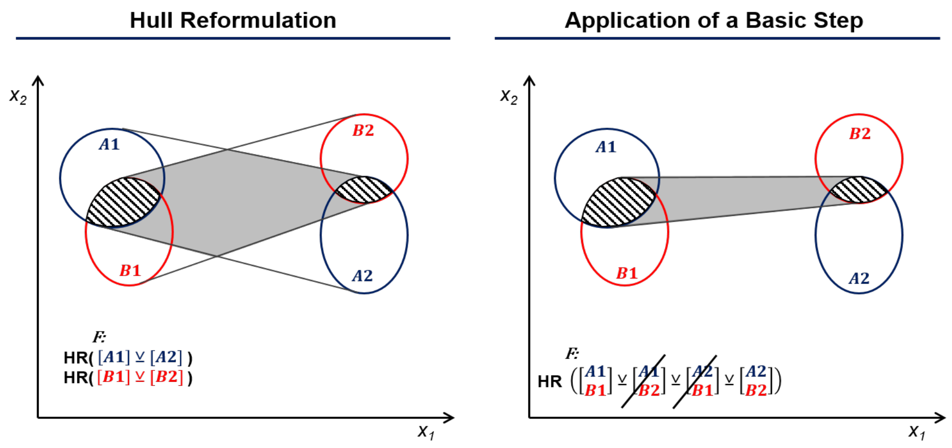
Hull Reformulation
Application of a Basic Step
x_2
x_1
$A1$
$B1$
$A2$
$B2$
F:
HR($[A1] \veebar [A2]$)
HR($[B1] \veebar [B2]$)
HR $\left(\begin{bmatrix}A1\\B1\end{bmatrix} \veebar \cancel{\begin{bmatrix}A1\\B2\end{bmatrix}} \veebar \cancel{\begin{bmatrix}A2\\B1\end{bmatrix}} \veebar \begin{bmatrix}A2\\B2\end{bmatrix}\right)$

Figure 1: Illustration of basic steps.

A basic step is an elementary, but fundamental, operation in disjunctive programming, tantamount in importance to a branching rule or cutting plane in mixed-integer programming. In words, a basic step applied to a disjunctive set $\mathcal{F} = \bigcap_{k\in\mathcal{K}} \bigcup_{i\in\mathcal{D}_k} \mathcal{S}_{ki}$ is the operation of taking two disjunctions p and q in $\mathcal{K}$ and bringing their intersection into disjunctive normal form. In set notation, a basic step takes some $p, q \in \mathcal{K} (p \neq q)$ and brings $\left(\cup_{i\in\mathcal{D}_p} \mathcal{S}_{pi}\right) \cap \left(\cup_{j\in\mathcal{D}_q} \mathcal{S}_{qj}\right)$ into DNF by replacing it with the set

$$\bigcup_{\substack{i\in\mathcal{D}_p\\ j\in\mathcal{D}_q}} \left(\mathcal{S}_{pi} \cap \mathcal{S}_{qj}\right) . \tag{6}$$

Since every application of a basic step replaces two disjunctions with a single disjunction, the set $\mathcal{F}$ can be brought into DNF by $K-1$ applications of basic steps. However, just as it is rarely necessary to find the integer hull of a mixed-integer program in order to solve it, one need not bring a disjunctive program into DNF to obtain useful bounds. In fact, a few judiciously chosen basic steps may suffice to adequately improve the lower bound. Figure 1 illustrates an application of a basic step between two disjunctions A and B, each possessing two disjuncts. The striped area delineates the true feasible region while the shaded (and striped) area marks the feasible region of the hull relaxation before and after the basic step. Clearly, the hull relaxation is much tighter after the basic step. Note that much of the gain is due to the fact that, after the basic step, two of the resulting disjuncts are empty sets and thus can be discarded.

From the above discussion, it should be clear that iteratively applying basic steps results in fewer disjunctions, but with certain disjunctions possibly possessing an exponential number of disjuncts. If we were to construct hull reformulations for these disjunctive programs as formulated in (5), which require disaggregated variables $\boldsymbol{\nu}_{ki}$ and y_{ki} for each disjunct, the resulting convex MINLP formulations would likely require the generation of additional decision variables after each basic step. Adding decision variables during the course of a branch-and-cut algorithm reeks havoc on the efficiency of most solvers, which helps explain why basic steps are not regularly used in mixed-integer programming computations.

Since basic steps have the potential to produce tighter relaxations, it is natural to ask: Which basic steps result in the tightest possible relaxation? Since an answer to this question was elusive at the time, Balas [3, Theorem 4.5] provided an answer to a simpler, but nonetheless important question: When will a basic step be ineffective? His result was specialized to the linear setting, but can be trivially extended to the convex setting below.

Proposition 1. *For $k = 1, 2$, let $\mathcal{F}_k = \cup_{i\in\mathcal{D}_k}\mathcal{C}_{ki}$, where each $\mathcal{C}_{ki}$ is a compact convex set. Then*

$$\mathcal{T}_L := \operatorname{clconv}(\mathcal{F}_1 \cap \mathcal{F}_2) = \operatorname{clconv}(\mathcal{F}_1) \cap \operatorname{clconv}(\mathcal{F}_2) =: \mathcal{T}_R \tag{7}$$

if and only if every extreme point of $\mathcal{T}_R$ is an extreme point of $\mathcal{C}_{1p} \cap \mathcal{C}_{2q}$ for some $(p, q) \in \mathcal{D}_1 \times \mathcal{D}_2$.

Unfortunately, evaluating this property is not practical in general, and can only be done tractably for some specific types of constraints. Meanwhile, for convex disjunctive programs, Ruiz and Grossmann [18] characterized additional types of constraints for which basic steps do not improve relaxations. Trespalacios and Grossmann [22] provided various heuristic rules concerning when to apply basic steps based on whether two disjunctions have variables in common, the number of disjuncts in a disjunction, and a parameter they define as the characteristic value of a disjunction, which represents how much the objective function changes if branched on the disjunction. The latter idea is similar to techniques used in strong/pseudocost/reliability branching.

3 Main results

In this section, we present our main result by answering the question: What is a bound improvement guarantee from a basic step? To accomplish this, we take advantage of a Lagrangian relaxation of a disaggregated convex disjunctive program, for which the linear case was originally considered in Trespalacios and Grossmann [23]. The review work by Guignard [14] discusses how Lagrangian relaxation can be used in different solution methods and applications. In Section 3.1, along with answering the above question, we introduce the central tool of this work, which we call a pseudo basic step. Section 3.2 then presents an illustrative example as well as insight into why pseudo basic steps may be less productive than actual basic steps. Discussion to contextualize pseudo basic steps follows in Section 3.3.

3.1 Partition relaxation and guaranteed bound improvements from basic steps

Consider a partial Lagrangian relaxation of a disaggregated convex disjunctive program (3) described by the set $\mathcal{K} = \{1, \dots, K\}$ of disjunctions:

$$\begin{aligned} LR_{\mathcal{K}}(\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_K) \quad &= \quad \min \quad (\mathbf{c} - \sum_{k\in\mathcal{K}} \boldsymbol{\lambda}_k)^\top \mathbf{x} + \sum_{k\in\mathcal{K}} \boldsymbol{\lambda}_k^\top \mathbf{v}_k && \text{(8a)} \\ &\qquad \text{s.t.} \quad \mathbf{v}_k \in \mathcal{F}_k = \bigcup_{i\in\mathcal{D}_k} \mathcal{C}_{ki} \qquad \forall\, k \in \mathcal{K} && \text{(8b)} \\ &\qquad\qquad \mathbf{x} \in \mathbb{R}^n \ . && \text{(8c)} \end{aligned}$$

The qualifier "partial" serves to emphasize that, while there may be other inequalities in an algebraic representation of the sets $\text{clconv}(\mathcal{F}_k)$ that could be dualized, we only relax the constraints $\mathbf{x} = \mathbf{v}_k$ linking the original decision vector and its copies. Moreover, consider the Lagrangian dual problem

$$z_{\mathcal{K}}^{\text{LD}} = \max \{LR_{\mathcal{K}}(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K) : \boldsymbol{\lambda}_k \in \mathbb{R}^n \ \forall k \in \mathcal{K}\} \ . \tag{9}$$

Proposition 2. *If optimal multipliers $\boldsymbol{\lambda}_1^*, \ldots, \boldsymbol{\lambda}_K^*$ exist for* (9)*, they must satisfy the condition $\sum_{k\in\mathcal{K}} \boldsymbol{\lambda}_k^* = \mathbf{c}$.*

Proof. If $\sum_{k\in\mathcal{K}} \boldsymbol{\lambda}_k \neq \mathbf{c}$, the minimization problem (8) is unbounded as $\mathbf{x}$ is unrestricted. $\square$

Henceforth, we only consider multipliers $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K$ such that $\sum_{k\in\mathcal{K}} \boldsymbol{\lambda}_k = \mathbf{c}$. Thus, the Lagrangian relaxation problem (8) can be further decomposed into K subproblems, each one being a convex disjunctive program in DNF, as follows:

$$LR_{\mathcal{K}}(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K) = \sum_{k\in\mathcal{K}} \min \left\{\boldsymbol{\lambda}_k^\top \mathbf{v} : \mathbf{v} \in \mathcal{F}_k\right\} \ . \tag{10}$$

Re-stating the Lagrangian relaxation as in (10) reveals that both formulations (8) and (10) are also partial Lagrangian relaxations of the hull relaxation (4), a relationship captured in the following proposition.

Proposition 3. *The hull relaxation* (4) *and the Lagrangian dual problem* (9) *are weak duals of one another, i.e., $z_{\mathcal{K}}^{Hrel} \geq z_{\mathcal{K}}^{LD}$. Moreover, if optimal multipliers $\boldsymbol{\lambda}_1^*, \ldots, \boldsymbol{\lambda}_K^*$ exist to* (4)*, then they are optimal for* (9) *and $z_{\mathcal{K}}^{Hrel} = z_{\mathcal{K}}^{LD}$.*

Proof. The weak duality claim follows from the fact that each subproblem $\min \left\{\boldsymbol{\lambda}_k^\top \mathbf{v} : \mathbf{v} \in \mathcal{F}_k\right\}$ in (10) is the minimization of a linear function over the union of compact convex sets, which is equivalent to minimizing the same linear function over the convex closure of the union of these sets (see, e.g., [18, Section 2.5]). Meanwhile, the existence of optimal multipliers $\boldsymbol{\lambda}_1^*, \ldots, \boldsymbol{\lambda}_K^*$ to the hull relaxation (4) implies, by [9, Definition 5.1.1], that $z_{\mathcal{K}}^{\text{Hrel}} = LR(\boldsymbol{\lambda}_1^*, \ldots, \boldsymbol{\lambda}_K^*)$. Hence, $z_{\mathcal{K}}^{\text{Hrel}} = z_{\mathcal{K}}^{\text{LD}}$ and $\boldsymbol{\lambda}_1^*, \ldots, \boldsymbol{\lambda}_K^*$ are optimal for (9) (see also [9, Proposition 5.1.4]). $\square$

Note that the hull relaxation (4) is a convex optimization problem, which means that Lagrange multipliers are typically only guaranteed to exist at a primal optimum under some form of constraint qualification. For example, Ceria and Soares [11] state that if the weak Slater condition holds for every disjunct in a convex disjunctive program in DNF, then the existence of multipliers at a primal optimum is guaranteed. For our purposes, we are more concerned with how to use multipliers if they exist than with the conditions guaranteeing their existence.

Let $\mathcal{P} = \{\mathcal{J}_1, \ldots, \mathcal{J}_P\}$ be a partition of $\mathcal{K}$, i.e., the sets $\mathcal{J}_p$ are non-empty disjoint subsets of $\mathcal{K}$ such that $\cup_{p=1}^P \mathcal{J}_p = \mathcal{K}$. For any partition $\mathcal{P}$ of $\mathcal{K}$ and any set of vectors $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_P$, we define the *partition relaxation* as

$$L_{\mathcal{P}}(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_P) = \sum_{p=1}^{P} \min \left\{\boldsymbol{\mu}_p^\top \mathbf{v} : \mathbf{v} \in \cap_{k\in\mathcal{J}_p} \mathcal{F}_k\right\} \ . \tag{11}$$

Each minimization problem (or *subproblem*) in the summation of (11) is a convex disjunctive program and can be solved as a convex MINLP, e.g., using a formulation akin to (5) or with a Big-M formulation as described in [13]. The next proposition states that, given any partition $\mathcal{P}$ of $\mathcal{K}$, one can simply sum the associated multipliers and expect a bound improvement, or at least no deterioration, relative to the previous relaxation given by (8).

Proposition 4. *For any set of vectors* $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K$ *and any partition* $\mathcal{P}$ *of* $\mathcal{K}$*, we have*

$$L_{\mathcal{P}}\left(\sum_{j \in \mathcal{J}_1} \boldsymbol{\lambda}_j, \ldots, \sum_{j \in \mathcal{J}_P} \boldsymbol{\lambda}_j\right) \geq LR_{\mathcal{K}}(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K) \ . \tag{12}$$

Proof. Since, for any $\mathcal{J} \in \mathcal{P}$, we have $\cap_{j \in \mathcal{J}} \mathcal{F}_j \subseteq \mathcal{F}_k$ for all $k \in \mathcal{J}$, it follows that

$$\sum_{p=1}^{P} \min \left\{ \sum_{k \in \mathcal{J}_p} \boldsymbol{\lambda}_k^\top \mathbf{v} : \mathbf{v} \in \cap_{k \in \mathcal{J}_p} \mathcal{F}_k \right\} \geq \sum_{p=1}^{P} \sum_{k \in \mathcal{J}_p} \min \left\{ \boldsymbol{\lambda}_k^\top \mathbf{v} : \mathbf{v} \in \mathcal{F}_k \right\} \ .$$

□

Assuming optimal multipliers to the hull relaxation (4) exist, Propositions 3 and 4 lead immediately to a computable bound on a simpler operation: the bound improvement from a basic step between two disjunctions k and l.

Corollary 1. *Let* $\boldsymbol{\lambda}_1^*, \ldots, \boldsymbol{\lambda}_K^*$ *be optimal multipliers to the hull relaxation* (4) *with* K *disjunctions. Without loss of generality, assume* $k = K - 1$ *and* $l = K$*. Then, the bound improvement* $\Delta(k, l)$ *from a basic step between two disjunctions* k *and* l *satisfies:*

$$\Delta(k, l) \geq L_{\{\{1\}, \ldots, \{K-2\}, \{K-1, K\}\}}\left(\boldsymbol{\lambda}_1^*, \ldots, \boldsymbol{\lambda}_{K-2}^*, \boldsymbol{\lambda}_{K-1}^* + \boldsymbol{\lambda}_K^*\right) - LR_{\mathcal{K}}(\boldsymbol{\lambda}_1^*, \ldots, \boldsymbol{\lambda}_K^*) \geq 0 \ . \tag{13}$$

Given multipliers $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K$ and a subset $\mathcal{J}$ of disjunctions, we refer to the solution of the convex disjunctive program

$$\min \left\{ \sum_{k \in \mathcal{J}} \boldsymbol{\lambda}_k^\top \mathbf{v} : \mathbf{v} \in \cap_{k \in \mathcal{J}} \mathcal{F}_k \right\} \tag{14}$$

as a *pseudo basic step* with respect to $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K$ and $\mathcal{J}$. Alternatively, one can see that a pseudo basic step is the solution of one subproblem in the partition relaxation (11). The qualifier "pseudo" is meant to distinguish it from an actual basic step. To be precise, recall that for an actual basic step involving a subset $\mathcal{J}$ of disjuncts, one has to perform the following operations to obtain a tighter relaxation: (a) intersect all disjunctions in $\mathcal{J}$; (b) bring their intersection into DNF, possibly creating a new disjunction with many disjuncts; and (c) solve the hull relaxation of the resulting convex disjunctive program. As discussed in Section 2, these steps are computationally expensive because the resulting hull reformulation, typically resembling formulation (5), may have many more decision variables after steps (a) and (b) are carried out. A second deficiency of an actual basic step is that, in step (c), all discrete features of the original disjunctive program are relaxed as one optimizes over the convex closure of all disjunctions. In contrast, a pseudo basic step retains some of the discrete features as it does not convexify the disjunctions in $\mathcal{J}$. On the other hand, one can see that an advantage of an actual basic step is that the relaxation in step (c) has a more "global" view since it takes into account the interaction of all disjunctions simultaneously, not just those in $\mathcal{J}$. A pseudo basic step attempts to achieve this "global" view through multipliers (which are not required in an actual basic step), but without having to consider all disjunctions simultaneously. Note that we do not require a pseudo basic step to use optimal multipliers to the hull relaxation (4). Finally, our definition assumes that (14) is solved to provable optimality. This need not be the case; one could prematurely terminate the solution of (14) and take the best available bound leading to a *suboptimal* pseudo basic step.

Given a partition $\mathcal{P} = \{\mathcal{J}_1, \dots, \mathcal{J}_P\}$ of $\mathcal{K}$, let $\mathcal{Q} = \{1, \dots, P\}$ be the set of disjunctions, each in DNF, resulting from sequential basic steps of the elements in each $\mathcal{J}_q$ for $q \in \mathcal{Q}$. That is, for each $q \in \mathcal{Q}$, $\mathcal{F}_q = \cap_{k \in \mathcal{J}_q} \mathcal{F}_k$ expressed in DNF. Thus, we can speak of the convex disjunctive program defined by $\mathcal{P}$ resulting from a sequence of actual basic steps:

$$\min \quad \mathbf{c}^\top \mathbf{x} \tag{15a}$$

$$\text{s.t.} \quad \mathbf{x} = \mathbf{v}_q \qquad \forall\, q \in \mathcal{Q} \quad (\boldsymbol{\mu}_q \in \mathbb{R}^n) \tag{15b}$$

$$\mathbf{v}_q \in \mathcal{F}_q \qquad \forall\, q \in \mathcal{Q}\ . \tag{15c}$$

The following corollary places guaranteed upper and lower bounds on one or more pseudo basic steps and, thus, summarizes where pseudo basic steps reside in the hierarchy of relaxations proposed by Balas [3].

Corollary 2. *Let $\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_K$ be a set of Lagrange multiplier vectors such that $\sum_{k \in \mathcal{K}} \boldsymbol{\lambda}_k = \mathbf{c}$. Let $\mathcal{P} = \{\mathcal{J}_1, \dots, \mathcal{J}_P\}$ be a partition of $\mathcal{K}$ and $\mathcal{Q} = \{1, \dots, P\}$ be the set of disjunctions, each in DNF, resulting from sequential basic steps of the elements in each $\mathcal{J}_q$ for $q \in \mathcal{Q}$. Then, we have*

$$z^* = L_{\{\mathcal{K}\}}\left(\sum_{k \in \mathcal{K}} \boldsymbol{\lambda}_k\right) \geq z_{\mathcal{Q}}^{Hrel} \geq z_{\mathcal{Q}}^{LD} \geq L_{\mathcal{P}}\left(\sum_{j \in \mathcal{J}_1} \boldsymbol{\lambda}_j, \dots, \sum_{j \in \mathcal{J}_P} \boldsymbol{\lambda}_j\right) \geq LR_{\mathcal{K}}(\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_K)\ , \tag{16}$$

with $z_{\mathcal{Q}}^{Hrel} = z_{\mathcal{Q}}^{LD}$ if optimal multipliers $\boldsymbol{\mu}_1^*, \dots, \boldsymbol{\mu}_P^*$ exist to the hull relaxation of (15).

Proof. The equality follows from the definition of the partition relaxation (11) and the assumption $\sum_{k \in \mathcal{K}} \boldsymbol{\lambda}_k = \mathbf{c}$, and states the obvious: the optimal value z^* of (3) is equal to the trivial partition relaxation when $\mathcal{P} = \{\mathcal{K}\}$. The first inequality follows from the fact that any hull relaxation of the original problem, regardless of the number of basic steps applied, is still a relaxation. The second inequality and the final assertion on conditions when $z_{\mathcal{Q}}^{\text{Hrel}} = z_{\mathcal{Q}}^{\text{LD}}$ follow from Proposition 3. The third inequality is due to the fact that the multipliers $\left(\sum_{j \in \mathcal{J}_1} \boldsymbol{\lambda}_j, \dots, \sum_{j \in \mathcal{J}_P} \boldsymbol{\lambda}_j\right)$ are suboptimal to the Lagrangian dual problem associated with (15). The last inequality is from Proposition 4. $\square$

The fact that $z_{\mathcal{Q}}^{\text{Hrel}} \geq L_{\mathcal{P}}\left(\sum_{j \in \mathcal{J}_1} \boldsymbol{\lambda}_j, \dots, \sum_{j \in \mathcal{J}_P} \boldsymbol{\lambda}_j\right)$ in Corollary 2 implies that a pseudo basic step can never improve the bound more than an actual basic step, i.e., taking an actual basic step and solving the corresponding hull relaxation. Meanwhile, since a pseudo basic step is meant to approximate an actual basic step, it is tempting to think that pseudo basic steps may always be inferior to actual basic steps in terms of bound improvement. The fact that $z^* = L_{\{\mathcal{K}\}}\left(\sum_{k \in \mathcal{K}} \boldsymbol{\lambda}_k\right)$ in Corollary 2, although obvious given the definition of a partition relaxation, reveals that in the extreme case a pseudo basic step recovers the optimal objective function value just as repeated applications of actual basic steps would do.

3.2 Illustrative example

We now present a simple convex disjunctive program in two variables to illustrate the difference in the relaxation improvement between actual and pseudo basic steps. In light of Corollary 2, which provides bounds on the improvement of a pseudo basic step, the purpose of this example is to (1) demonstrate how accurate these bounds may be, and (2) explain what goes wrong when a pseudo basic step does not achieve the same improvement as an actual basic step. Consider the following example, depicted graphically in

Figure 2a:

$$\begin{aligned}
\min \quad & 0.2x_1 + x_2 && \text{(17a)}\\
\text{s.t.} \quad & \mathcal{F}_1 = \left[(x_1)^2 + \tfrac{1}{4}(x_2-5)^2 \le 1\right] \vee \left[(x_1-5)^2 + \tfrac{1}{4}(x_2-2)^2 \le 1\right] && \text{(17b)}\\
& \mathcal{F}_2 = \left[(x_1)^2 + \tfrac{1}{4}(x_2-2)^2 \le 1\right] \vee \left[(x_1-5)^2 + \tfrac{1}{4}(x_2-5)^2 \le 1\right] && \text{(17c)}\\
& \mathcal{F}_3 = \left[(x_1)^2 + \tfrac{1}{4}(x_2-3.5)^2 \le 1\right] \vee \left[(x_1-5)^2 + \tfrac{1}{4}(x_2-3.5)^2 \le 1\right]. && \text{(17d)}
\end{aligned}$$

The first disjunction (17b) is represented in Figure 2a by the ellipses with a continuous thin line, the second (17c) with a dashed line, and the third (17d) with a thick continuous line. The feasible region is shown in the striped area. The direction of the objective function is also shown in Figure 2a with an arrow, and the optimal solution is shown with a dot. Disaggregating the problem as in (3) and using a compact notation,

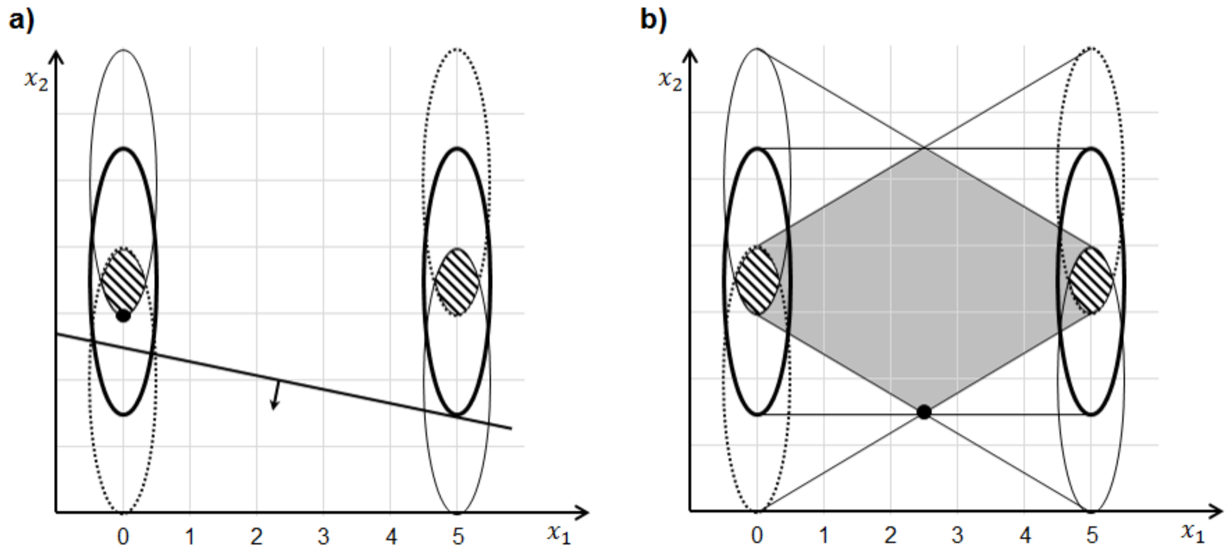

Figure 2: Illustration of example (17). a) Shows the feasible region and optimal solution. b) Shows the continuous relaxation of the hull reformulation, projected onto the original space.

we represent problem (17) as follows:

$$\begin{aligned}
\min \quad & 0.2x_1 + x_2 && && \text{(18a)}\\
\text{s.t.} \quad & x_j = v_{kj} && \forall\, k \in \{1,2,3\}, j \in \{1,2\} \quad (\lambda_{kj} \in \mathbb{R}) && \text{(18b)}\\
& \mathbf{v}_k \in \mathcal{F}_k && \forall\, k \in \{1,2,3\}, && \text{(18c)}
\end{aligned}$$

where j indexes the dimensions of $\mathbf{x}$. The partial Lagrangian relaxation of (18) is as follows:

$$\begin{aligned}
\min \quad & \left(0.2 - \sum_{k\in\{1,2,3\}} \lambda_{k1}\right) x_1 + \left(1 - \sum_{k\in\{1,2,3\}} \lambda_{k2}\right) x_2 + \sum_{k\in\{1,2,3\}} (\lambda_{k1} v_{k1} + \lambda_{k2} v_{k2}) && \text{(19a)}\\
\text{s.t.} \quad & \mathbf{v}_k \in \mathcal{F}_k \qquad \forall\, k \in \{1,2,3\}. && \text{(19b)}
\end{aligned}$$

In order to obtain the optimal multipliers that maximize (19), the hull relaxation of (18) is solved. Note that Proposition 2 asserts that any multipliers that maximize (19) will satisfy $0.2 = \sum_{k\in\{1,2,3\}} \lambda_{k1}$ and

Constraint	$\boldsymbol{\lambda}_1^*$	$\boldsymbol{\lambda}_2^*$	$\boldsymbol{\lambda}_3^*$
$x_1 - v_{k1} = 0$	0.200	0	0
$x_2 - v_{k2} = 0$	0.334	0	0.666

Table 1: Lagrange multipliers of the hull relaxation of (17).

$1 = \sum_{k\in\{1,2,3\}} \lambda_{k2}$, while Proposition 3 implies that (19) is a Lagrangian relaxation of both (18) and its hull relaxation (20). The hull relaxation of (17) can be expressed as follows:

$$\begin{aligned}
\min \quad & 0.2x_1 + x_2 && && \text{(20a)}\\
\text{s.t.} \quad & x_j = v_{kj} && \forall\ k \in \{1,2,3\}, j \in \{1,2\} \quad (\lambda_{kj} \in \mathbb{R}) && \text{(20b)}\\
& \mathbf{v}_k \in \operatorname{clconv}(\mathcal{F}_k) && \forall\ k \in \{1,2,3\}, && \text{(20c)}
\end{aligned}$$

where the convex hull of each disjunction can be expressed in algebraic form using the perspective function and the rotated second-order cone constraint described in the introduction. For example, clconv $(\mathcal{F}_1)$ of (20) can be represented as follows ($k = 1$ below):

$$\begin{aligned}
& v_{k1} = \nu_{k11} + \nu_{k21} && && \text{(21a)}\\
& v_{k2} = \nu_{k12} + \nu_{k22} && && \text{(21b)}\\
& t_{k11} + \tfrac{1}{4}t_{k12} - \tfrac{5}{2}\nu_{k12} + \tfrac{21}{4}y_{k1} \le 0 && && \text{(21c)}\\
& t_{k21} - 10\nu_{k21} + \tfrac{1}{4}t_{k22} - \nu_{k22} + 25y_{k2} \le 0 && && \text{(21d)}\\
& (\nu_{kij})^2 \le t_{kij}y_{ki} && \forall j \in \{1,2\}, i \in \{1,2\} && \text{(21e)}\\
& L_{kj}y_{ki} \le \nu_{kij} \le U_{kj}y_{ki} && \forall j \in \{1,2\}, i \in \{1,2\} && \text{(21f)}\\
& y_{k1} + y_{k2} = 1 && && \text{(21g)}\\
& t_{kij} \ge 0 && \forall j \in \{1,2\}, i \in \{1,2\} && \text{(21h)}\\
& y_{k1}, y_{k2} \ge 0\ , && && \text{(21i)}
\end{aligned}$$

where L_{kj} and U_{kj} are easily computed lower and upper bounds on ν_{kij}. A similar representation can be applied to disjunctions 2 and 3. Note that the final algebraic representation of (20) can be more compact after algebraic substitutions. The hull relaxation of (17), projected onto the original space, is presented in Figure 2b. The optimal solution is shown with a dot in Figure 2b. Table 1 lists the values of the optimal multipliers of (20) corresponding to constraints (20b). It is easy to see from these values (specifically, $\boldsymbol{\lambda}_2^* = \mathbf{0}$) that constraints (17c), corresponding to disjunction 2, are not active. The optimal value of (17) is 2.99, and the optimal value of its hull relaxation is 1.97. The hull relaxation of the problem after applying a basic step between disjunctions 1 and 2 is as follows:

$$\begin{aligned}
\min \quad & 0.2x_1 + x_2 && && \text{(22a)}\\
\text{s.t.} \quad & x_j = v_{kj} && \forall\ k \in \{1,2\}, j \in \{1,2\} \quad (\lambda_{kj} \in \mathbb{R}) && \text{(22b)}\\
& \mathbf{v}_1 \in \operatorname{clconv}(\mathcal{F}_1 \cap \mathcal{F}_2) && && \text{(22c)}\\
& \mathbf{v}_2 \in \operatorname{clconv}(\mathcal{F}_3)\,. && && \text{(22d)}
\end{aligned}$$

Selected disjunctions	Basic step	Pseudo basic step
1, 2	2.99	2.31
1, 3	2.99	2.99
2, 3	2.45	2.31

Table 2: Objective function value of the hull relaxation of (17) after intersecting disjunctions.

The problem resulting from a pseudo basic step between disjunctions 1 and 2 is as follows:

$$\min \quad (0.2+0)v_{11} + (0.334+0)v_{12} + 0v_{21} + 0.666v_{22} \tag{23a}$$

$$\text{s.t.} \quad \mathbf{v}_1 \in (\mathcal{F}_1 \cap \mathcal{F}_2) \tag{23b}$$

$$\mathbf{v}_2 \in \mathcal{F}_3 \ . \tag{23c}$$

Table 2 presents the objective function values of the hull relaxation after applying basic steps and pseudo basic steps. The table shows that any actual basic step and any pseudo basic step improve the hull relaxation (relative to that of the original problem, which has a value of 1.97). However, an actual basic step provides a larger improvement when applied to disjunction 1 and 2, and when applied to disjunction 2 and 3. In the first case, a basic step between disjunctions 1 and 2 yields an objective value of the relaxation that is the same as the optimal solution (which is 2.99), while the pseudo basic step gives a worse relaxation of 2.31. The illustration of the hull relaxation after basic steps, projected onto the original space, is presented in Figure 3. From the initial multipliers of the hull relaxation and Figure 3a, it is easy to see why a

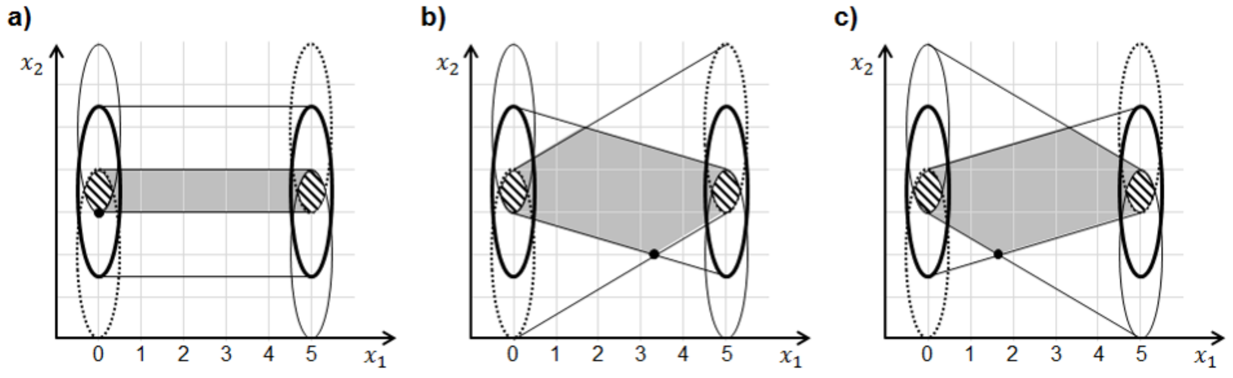

Figure 3: Illustration of the hull relaxation after basic steps for (17) intersecting a) disjunctions 1 and 2; b) disjunctions 1 and 3; and c) disjunctions 2 and 3

pseudo basic step between disjunctions 1 and 2 is inferior to an actual basic step. Figure 3a shows the basic step and it is clear that disjunction 3 is inactive. The optimal Lagrange multipliers corresponding to disjunction 3 are $\boldsymbol{\lambda}_3^* = [0,0]^\top$, while the optimal multipliers to the new disjunction are $\boldsymbol{\lambda}_{12}^* = \mathbf{c} = [0.2, 1]^\top$. Meanwhile, if a pseudo basic step is applied instead of an actual basic step, the obtained (suboptimal) multipliers are $\boldsymbol{\lambda}_3 = [0, 0.666]^\top$ and $\boldsymbol{\lambda}_{12} = [0.2, 0.334]^\top$. Here, both the optimal multipliers (obtained by solving the hull relaxation of the disjunctive program after applying a basic step) and suboptimal multipliers (obtained by adding the optimal multipliers $\boldsymbol{\lambda}_k^*$ associated with the disjunctions of the original problem) satisfy $\sum_{k\in\mathcal{K}} \boldsymbol{\lambda}_k = \mathbf{c}$. However, the suboptimal multipliers provide a weaker bound than the optimal ones as shown in Table 2 (the bound obtained using optimal multipliers is 2.99, but the one with suboptimal multipliers is 2.31). A similar observation of the basic step between disjunctions 1 and 3 reveals that the pseudo basic step yields the same optimal multipliers as the basic step ($\boldsymbol{\lambda}_2 = \boldsymbol{\lambda}_2^* = [0,0]^\top$ and $\boldsymbol{\lambda}_{13} = \boldsymbol{\lambda}_{13}^* = [0.2,1]^\top$). Finally, the multipliers of the pseudo basic step between disjunctions 2 and 3 are $\boldsymbol{\lambda}_1 = [0.2, 0.334]^\top$ and $\boldsymbol{\lambda}_{23} = [0, 0.666]^\top$. In contrast, the optimal multipliers when solving the hull relaxation corresponding to the application of an actual basic step are $\boldsymbol{\lambda}_1^* = [0.3, 0.501]^\top$ and $\boldsymbol{\lambda}_{23}^* = [-0.1, 0.499]^\top$.

3.3 Discussion

A potential drawback of pseudo basic steps is their reliance on Lagrange multipliers. Although Proposition 3 suggests an obvious way to obtain an initial set of multipliers, i.e., by solving the hull relaxation (4), for large convex disjunctive programs, it is typically more convenient computationally to remain in a lower-dimensional space using a Big-M formulation (see, e.g., [13]). As a consequence, one would see little value constructing the hull reformulation and solving its relaxation since it will not be used thereafter. That said, it is important to emphasize that multipliers can be obtained in three ways: 1) exactly, by solving the hull relaxation to provable optimality as a convex optimization problem in polynomial time; 2) approximately, subject to the condition $\sum_{k\in\mathcal{K}} \boldsymbol{\lambda}_k = \mathbf{c}$; or, in some cases, 3) analytically (see Section 4.2 below). In fact, there is no guarantee that optimal multipliers to the hull relaxation (4) will yield greater improvements than using a set of suboptimal multipliers. One could even use several sets of multipliers and apply pseudo basic steps using these different sets to improve the bound.

Algorithmically, the problem of choosing the disjunctions for a basic step (actual or pseudo) that will yield the most improvement could be performed in manner analogous to strong branching in mixed-integer programming. In "full" strong branching, all integer variables that take a fractional value at the current node being evaluated are "branched on" and the one yielding the largest improvement is chosen for branching [1]. To apply pseudo basic steps in convex disjunctive programming, one could evaluate all pairs of disjunctions, using the existing theory presented above to winnow out unproductive ones, and choose the one that yields the largest bound improvement. Just as full strong branching is powerful, but computationally expensive, we expect more heuristics to be developed to address this issue.

4 Computational experiments

In this section, we investigate two convex disjunctive programs to illustrate the improvements from pseudo basic steps and the partition relaxation. Although our results apply for many types of convex disjunctive programs, our computational experiments focus exclusively on MIQCP formulations to showcase the improvements possible over existing state-of-the-art solvers. While the two problems are different in several respects, the most important distinction for our purposes is how Lagrange multipliers are obtained since our approach relies on their availability. In Section 4.1 multipliers are computed by solving the hull relaxation, whereas in Section 4.2, we show that multipliers can be determined analytically without resorting to the hull reformulation. All experiments were carried out on a Dell Precision T7600 running two 64-bit 3.10 GHz Intel Xeon E5-2687W-0 processors with 64 GB of RAM.

			Basic step		Pseudo basic step			
Instance	Opt. val.	HR	relaxation	time (s)	relaxation	time (s)	parallel (s)	% improvement
1	-3,516	-4,736	-3,903	993.5	-4,295	6.5	4.9	52.9
2	-3,654	-4,939	-4,054	29.7	-4,362	3.7	1.9	65.2
3	-4,351	-4,868	-4,384	435.7	-4,418	5.0	2.6	93.1
4	-3,470	-4,820	-3,992	1472.4	-4,321	3.9	2.3	60.2
5	-5,835	-7,459	-6,453	534.2	-6,765	4.6	2.5	69.0
6	-4,527	-6,542	-5,431	15.1	-5,739	6.1	2.6	72.3
7	-3,377	-5,094	-3,441	154.0	-4,472	6.3	3.2	37.6
8	-3,251	-4,268	-3,538	200.5	-3,628	8.2	5.7	87.8
9	-3,622	-4,120	-3,976	75.6	-4,060	4.1	1.4	41.5
10	-3,764	-4,872	-3,931	113.1	-4,473	5.3	2.2	42.4

Table 3: Comparison of actual basic steps with pseudo basic steps.

4.1 Conic quadratic disjunctive programs

In this section, we numerically investigate the quality of pseudo basic steps relative to an actual basic step. We consider a set of randomly generated conic quadratic disjunctive programs having the following form:

$$\min \quad \mathbf{c}^\top \mathbf{x} \tag{24a}$$

$$\text{s.t.} \quad \mathbf{x} \in \bigcap_{k \in \mathcal{K}} \bigcup_{i \in \mathcal{D}_k} \mathcal{C}_{ki} \tag{24b}$$

where $\mathcal{C}_{ki} = \left\{ \mathbf{x} \in \mathbb{R}^n : \mathbf{x}^\top \mathbf{Q}_{ki} \mathbf{x} + \mathbf{b}_{ki}^\top \mathbf{x} + \gamma_{ki} \leq 0 \right\}$. This section presents the results of two experiments with different sets of random instances of conic quadratic disjunctive programs, each set with 10 instances. The main objective of the first experiment is to show the performance of pseudo basic steps vs. actual basic steps. This set contains small instances, since solving the hull relaxation after a few basic steps becomes intractable in larger problems. All of the instances of set 1 can be easily solved by commercial solvers. The second experiment contains larger instances that cannot be solved by CPLEX in 10 minutes. The objective in this experiment is to compare the lower bound obtained using random partitions of the problem against the lower bound of CPLEX after reaching the time limit of ten minutes. For all instances, the variables have a lower bound of -100 and upper bound of 100. The cost coefficient of every variable is a random number between -1000 and 1000. For each disjunct ki and each variable j, the coefficients $[q_{kijj}, b_{kij}, \gamma_{ki}]$ are random numbers between $[0.1, -1, 0]$ and $[1, 1, 2]$. The 10 instances of the first experiment have 8 disjunctions, with 5 disjuncts each, and $\mathbf{x} \in \mathbb{R}^{10}$. The algorithm to randomly generate basic steps enforces that at least 4 basic steps are applied.

Table 3 presents the performance of actual basic steps vs. pseudo basic steps for these randomly generated instances. It shows that the improvement of a pseudo basic step for these instances is between 40% to 90%. This result shows how much of the bound improvement of the basic step was captured by the pseudo basic step. The value is calculated by dividing the absolute difference between the relaxation after the pseudo basic step and the hull relaxation (column HR in Table 3) by the absolute difference between the relaxation after the actual basic step and the hull relaxation. The optimal multipliers for the Lagrangian dual of the problem, before basic steps, were computed by solving the hull relaxation. This is included in the solution

		CPLEX (600 s)		TresPapas		
Instance	Best known	BM LB	HR LB	LB	time (s)	parallel (s)
1	-7,653	-16,933	-20,590	-14,371	234.1	110.3
2	-6,595	-15,498	-16,638	-10,195	265.8	257.5
3	-6,691	-17,731	-17,858	-12,742	136.9	95.9
4	-6,447	-21,127	-16,897	-12,545	111.1	69.4
5	-6,482	-17,071	-19,300	-12,298	328.7	176.0
6	-7,133	-20,026	-20,070	-12,660	210.6	102.2
7	-6,391	-18,289	-18,191	-12,990	214.1	73.8
8	-5,065	-15,215	-14,140	-9,041	160.1	109.9
9	-5,971	-18,202	-16,570	-10,190	150.9	67.4
10	-6,795	-27,926	NA	-12,246	294.9	170.1

Table 4: Comparison of lower bound for TresPapas algorithm against the lower bound of the Big-M (BM) and Hull Reformulation (HR) models after 10 minutes using CPLEX.

time for the pseudo basic step results, both in serial and parallel. As expected, Table 3 shows that the relaxation after basic steps is better than the bound associated with the corresponding pseudo basic steps. However, it is important to note the required time to solve the NLP after applying multiple basic steps. For example, instance 4 requires about 25 minutes to solve the NLP and to provide a lower bound to the problem. In contrast, all of the pseudo basic steps can be computed in less than 10 seconds, and even faster when run in parallel.

The second set involves instances with 15 disjunctions, with 10 disjuncts each, and $\mathbf{x} \in \mathbb{R}^{30}$. The algorithm to randomly generate basic steps enforces that at least 11 basic steps are applied. In this case, the randomization algorithm also enforces that at most 5 disjunctions are intersected (i.e. the resulting disjunctive program will have either 3 or 4 disjunctions. If it has 3 disjunctions, each of these will be the result of intersecting 5 of the original disjunctions. If it has 4 disjunctions, any resulting disjunction is the intersection of at most 5 of the original disjunctions).

Table 4 presents the lower bound provided by the algorithm (named TresPapas), as well as the lower bound for the hull formulation and Big-M after 10 minutes using CPLEX 12.6.3. In Table 4, the optimal multipliers for the Lagrangian dual of the original problems were computed by solving the hull relaxation. This is included in the solution time for the pseudo basic step results, both in serial and parallel. Table 4 shows that the lower bound using TresPapas is much better than the lower bound provided by CPLEX after 10 minutes. Furthermore, 9 of the 10 instances finish in less than 5 minutes when run in serial, and in less than 2 minutes when run in parallel.

4.2 K-means clustering

The K-means clustering problem is a widely solved problem in unsupervised learning and one of the most basic forms of clustering. It is typically solved heuristically without any regard for a provable lower bound. In this section, we show that (1) state-of-the-art MIP solvers struggle to solve these problems to provable optimality as well as to close the optimality gap when given a near optimal solution, and (2) our partition relaxation can be much more effective at closing the gap.

The K-means clustering problem is defined as follows. Suppose we are given N points $\mathbf{p}_1, \ldots, \mathbf{p}_N$ such that $\mathbf{p}_i \in \mathbb{R}^D$, an integer $K \in \mathcal{N} = \{1, \ldots, N\}$, and a distance metric $\mathbb{D} : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$. Our goal is to determine K clusters such that each point $\mathbf{p}_i$ is assigned to a cluster with a centroid $\mathbf{c}_k \in \mathbb{R}^D$ for $k \in \mathcal{K} = \{1, \ldots, K\}$ and the total distance of points to centroids is minimized. Not only does this problem have a natural disjunctive formulation (a point must be assigned to cluster 1 or cluster 2 or ... cluster K), it also has a remarkably simple structure. It seems reasonable to expect state-of-the-art MIP solvers to be able to exploit this structure if we expect them to handle more difficult problems possessing embedded clustering decisions. Indeed, many problems in location theory involve some sort of clustering component. For example, the problem of determining where K emergency facilities should be located to best serve a city in the wake of a natural disaster involves clustering decisions.

4.2.1 MIQCP formulations

When cast as a convex disjunctive program, the problem of finding K optimal clusters takes the form

$$\min \sum_{i \in \mathcal{N}} d_i \tag{25a}$$

$$\text{s.t.} \quad (\mathbf{c}, \mathbf{d}) \in \bigcap_{i \in \mathcal{N}} \bigcup_{k \in \mathcal{K}} \mathcal{C}_{ik} \tag{25b}$$

where $\mathcal{C}_{ik} = \big\{(\mathbf{c}, \mathbf{d}) \in \mathbb{R}^{K \times D} \times \mathbb{R}^N_+ : d_i \geq \mathbb{D}(\mathbf{p}_i, \mathbf{c}_k)\big\}$. Note that there exist N disjunctions, one for each point $i \in \mathcal{N}$, and K disjuncts per disjunction. Note that the sets $\mathcal{C}_{ik}$ are unbounded in this form. Without loss of generality, we assume that all points $\mathbf{p}_1, \ldots, \mathbf{p}_N$ have been normalized to reside in a D-dimensional hypercube and, thus, we may re-express $\mathcal{C}_{ik}$ as a compact set $\mathcal{C}_{ik} = \big\{(\mathbf{c}, \mathbf{d}) \in [0,1]^{K \times D} \times [0,1]^N : d_i \geq \mathbb{D}(\mathbf{p}_i, \mathbf{c}_k)\big\}$. Henceforth, we only consider the squared L_2 norm as the distance metric, i.e., $\mathbb{D}(\mathbf{p}_i, \mathbf{c}_k) = \sum_{j=1}^D (p_{ij} - c_{kj})^2$.

For each point-cluster pair, let y_{ik} be a binary decision variable taking value 1 if point i is assigned to cluster k and 0 otherwise. For each point $i \in \mathcal{N}$, define $M_i := \max\{\mathbb{D}(\mathbf{p}_i, \mathbf{p}_{i'}) : i' \in \mathcal{N}\}$. A straightforward Big-M MIQCP formulation is

$$\begin{aligned}
\min_{\mathbf{c},\mathbf{d},\mathbf{y}} \quad & \sum_{i \in \mathcal{N}} d_i && && \text{(26a)}\\
\text{s.t.} \quad & d_i \geq \sum_{j=1}^{D} (p_{ij} - c_{kj})^2 - M_i(1 - y_{ik}) && \forall\, i \in \mathcal{N}, k \in \mathcal{K} && \text{(26b)}\\
& \sum_{k \in \mathcal{K}} y_{ik} = 1 && \forall\, i \in \mathcal{N} && \text{(26c)}\\
& \mathbf{c}_k \in \mathbb{R}^D && \forall\, k \in \mathcal{K} && \text{(26d)}\\
& d_i \in \mathbb{R}_+ && \forall\, i \in \mathcal{N} && \text{(26e)}\\
& y_{ik} \in \{0,1\} && \forall\, i \in \mathcal{N}, k \in \mathcal{K} && \text{(26f)}
\end{aligned}$$

This is an extension of a typical K-medoids MILP formulation, which builds clusters using the L_1 distance metric, that one would find in an integer programming textbook (see, e.g., [15]). Clearly, if point i is assigned to cluster ($y_{ik} = 1$), then constraint (26b) will be tight and d_i will equal the squared Euclidean distance between point $\mathbf{p}_i$ and centroid $\mathbf{c}_k$. Otherwise, this constraint is loose. Constraints (26c) ensure that each point is assigned to exactly one cluster.

Now consider a hull reformulation, also an MIQCP formulation, of the K-means clustering problem. Let M be a large scalar.

$$
\begin{aligned}
\min_{\mathbf{c},\mathbf{c}',\mathbf{d},\mathbf{d}',\mathbf{t},\mathbf{y}} \quad & \sum_{i\in\mathcal{N}} d_i && && \text{(27a)}\\
\text{s.t.} \quad & d_i = \sum_{k\in\mathcal{K}} d'_{ik} && \forall\, i \in \mathcal{N} && \text{(27b)}\\
& c_{kj} = c'_{ki0j} + c'_{ki1j} && \forall\, k \in \mathcal{K}, i \in \mathcal{N}, j \in \{1,\dots,D\} && \text{(27c)}\\
& \sum_{j=1}^{D} \left(p_{ij}^2 y_{ik} - 2p_{ij}c'_{ki1j} + t_{kij}\right) - d'_{ik} \le 0 && \forall\, i \in \mathcal{N}, k \in \mathcal{K} && \text{(27d)}\\
& c'^2_{ki1j} \le t_{kij} y_{ik} && \forall\, i \in \mathcal{N}, k \in \mathcal{K}, j \in \{1,\dots,D\} && \text{(27e)}\\
& -M(1-y_{ik}) \le c'_{ki0j} \le M(1-y_{ik}) && \forall\, k \in \mathcal{K}, i \in \mathcal{N}, j \in \{1,\dots,D\} && \text{(27f)}\\
& \sum_{k\in\mathcal{K}} y_{ik} = 1 && \forall\, i \in \mathcal{N} && \text{(27g)}\\
& \mathbf{c}_k \in \mathbb{R}^D && \forall\, k \in \mathcal{K} && \text{(27h)}\\
& \mathbf{c}'_{kib} \in [-M, M]^D && \forall\, k \in \mathcal{K}, i \in \mathcal{N}, b \in \{0,1\} && \text{(27i)}\\
& d_i \in \mathbb{R} && \forall\, i \in \mathcal{N} && \text{(27j)}\\
& d'_{ik} \in \mathbb{R}_+ && \forall\, i \in \mathcal{N}, k \in \mathcal{K} && \text{(27k)}\\
& t_{kij} \in \mathbb{R}_+ && \forall\, i \in \mathcal{N}, k \in \mathcal{K}, j \in \{1,\dots,D\} && \text{(27l)}\\
& y_{ik} \in \{0,1\} && \forall\, i \in \mathcal{N}, k \in \mathcal{K} && \text{(27m)}
\end{aligned}
$$

Constraints (27d) and (27e) ensure that when point i is assigned to cluster k, d'_{ik} takes the correct nonnegative value. Note that, in order to maintain MIQCP representability, the auxiliary decision variable t_{kij} is used in constraints (27d), instead of the term $\frac{c'^2_{ki1j}}{y_{ik}}$, along with the rotated second-order cone constraints (27e). Constraints (27f) ensure that when point i is not assigned to cluster k ($y_{ik} = 0$), the centroid copy $\mathbf{c}'_{ki0} = \mathbf{0}$; otherwise, $\mathbf{c}'_{ki0} \in [-M, M]^D$ so that $c_{kj} = c'_{ki0j} + c'_{ki1j}$ can hold for every k and i.

It is possible to impose tighter variable bounds, e.g., requiring $\mathbf{c}_k \in [0,1]^D$, $\mathbf{c}'_{kib} \in [0,1]^D$, $d_i \in [0,1]$, and $d'_{ik} \in [0,1]$, as well as modifying constraints (27f) to read $c'_{ki0j} \le (1-y_{ik})$ (the constraint $-M(1-y_{ik}) \le c'_{ki0j}$ vanishes when $\mathbf{c}'_{kib} \in [0,1]^D$). While we use the tighter bounds in computational experiments, formulation (27) makes the proof in Proposition 5 somewhat simpler.

Several remarks are in order.

Remark 2. Both the Big-M formulation (26) and hull reformulation (27) are expressed with a so-called unary representation. One may also express constraints (26c) as SOS1 constraints or using a binary representation [24].

Remark 3. Suppose $(\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_K) = (\mathbf{c}_1^*, \mathbf{c}_2^*, \dots, \mathbf{c}_K^*)$ in an optimal solution. Then, any permutation $(k_1, k_2, \dots, k_K)$ of the indices of $\mathcal{K}$ such that $(\mathbf{c}_{k_1}, \mathbf{c}_{k_2}, \dots, \mathbf{c}_{k_K}) = (\mathbf{c}_1^*, \mathbf{c}_2^*, \dots, \mathbf{c}_K^*)$ is also part of an optimal solution. To eliminate these equivalent optima, one can add symmetry breaking constraints

$$c_{k-1,1} \le c_{k,1}, \forall\, k \in \mathcal{K} \setminus \{1\} \;, \tag{28}$$

which require the centroids to be ordered in ascending order according to the first index/dimension. Clearly, any index/dimension could be selected.

Remark 4. Assuming all points are distinct, one can also add the constraints

$$\sum_{i\in\mathcal{N}} y_{ik} \geq 1 \qquad \forall k \in \mathcal{K} \tag{29}$$

to ensure that each cluster contains at least one point.

4.2.2 Partition relaxation

Consider a partial Lagrangian relaxation of the hull reformulation (27) in which constraints (27b) and (27c) are dualized with multipliers λ_i and $\boldsymbol{\mu}_{ki}$, respectively. This leads to the Lagrangian dual problem

$$\max\{LR_{\mathcal{N}}(\boldsymbol{\lambda},\boldsymbol{\mu}) : (\boldsymbol{\lambda},\boldsymbol{\mu}) \in \mathbb{R}^N \times \mathbb{R}^{K\times N\times D}\} \tag{30}$$

where

$$\begin{aligned} LR_{\mathcal{N}}(\boldsymbol{\lambda},\boldsymbol{\mu}) = \min \quad & \sum_{i\in\mathcal{N}}(1-\lambda_i)d_i + \sum_{i\in\mathcal{N}}\lambda_i\left(\sum_{k\in\mathcal{K}} d'_{ik}\right) \\ & + \sum_{k\in\mathcal{K}}\left(\mathbf{0} - \sum_{i\in\mathcal{N}}\boldsymbol{\mu}_{ik}\right)^\top \mathbf{c}_k + \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{N}}\boldsymbol{\mu}_{ik}^\top\left(\sum_{b\in\{0,1\}}\mathbf{c}'_{kib}\right) & \text{(31a)} \\ \text{s.t.} \quad & \text{(27d)} - \text{(27m)} & \text{(31b)} \end{aligned}$$

The next proposition shows that, for the K-means clustering hull reformulation (27), optimal multipliers $\boldsymbol{\lambda}^*$and $\boldsymbol{\mu}^*$ to the Lagrangian dual problem (30), or equivalently, to the dual of the continuous relaxation of (27), can be computed analytically.

Proposition 5. *In any optimal solution to the Lagrangian dual problem* (30)*, we have* $\lambda_i^* = 1$ *for all* $i \in \mathcal{N}$ *and* $\boldsymbol{\mu}_{ki}^* = \mathbf{0}$ *for all* $k \in \mathcal{K}, i \in \mathcal{N}$.

Proof. Let $\text{sign}(a) = +1$ if $a \geq 0$ and -1 otherwise. Since d_i is unconstrained in the Lagrangian relaxation problem (31), it is clear that $1 - \lambda_i^* = 0$ for all $i \in \mathcal{N}$, otherwise the minimization is unbounded as $d_i^* \to -\text{sign}(1-\lambda_i^*)\infty$ for some $i \in \mathcal{N}$. Now suppose $\mu^*_{\hat{k}\hat{i}\hat{j}} \neq 0$ for some $(\hat{k},\hat{i},\hat{j})$. Then, the minimization (31) is essentially unbounded with a solution having $c'^*_{\hat{k}\hat{i}0\hat{j}} \to -\text{sign}(\mu^*_{\hat{k}\hat{i}\hat{j}})M$ and $y^*_{\hat{i}\hat{k}} = 0$. In words, point $\hat{i}$ will be assigned to some other cluster/disjunct $k' \neq \hat{k}$ so that the centroid copy $\mathbf{c}'_{\hat{k}\hat{i}0}$ associated with not selecting cluster $\hat{k}$ can go to plus or minus infinity (or, in this formulation, some large scalar M). Meanwhile, $y^*_{\hat{k}\hat{i}} = 0$ and the corresponding rotated cone constraint (27e) together imply that $c'^*_{\hat{k}\hat{i}1\hat{j}} = 0$. Since we are minimizing and $d'_{\hat{i}\hat{k}} \geq 0$, it follows that $d'^*_{\hat{i}\hat{k}} = 0$ in constraint (27d) showing that the objective function term $\sum_{i\in\mathcal{N}}\lambda_i\sum_{k\in\mathcal{K}} d'_{ik}$ is not affected by increasing M. □

We consider two ways of constructing a partition relaxation. Algorithm 1 describes a deterministic heuristic that relies on a primal solution. Essentially, it attempts to distribute points assigned to the same cluster in the given primal solution to different subproblems. Recall that a subproblem refers to one of the minimization problems in the partition relaxation (11). The rationale for such point-subproblem assignments is to force points that are not in the same cluster, and thus likely to be farther in distance from one another, to be clustered together in a subproblem and consequently increase the lower bound. Algorithm 2 describes a randomized approach to partitioning points in which each subproblem has the same number of points. We performed limited experimentation in which subproblems could have a different number of points, but did not observe much of a benefit and so restricted ourselves to the current setting. Randomizing the partition allows us to see if there is any benefit to using a deterministic approach and to determine what might happen if the problems are solved with a naive parallel implementation.

Algorithm 1 Deterministic heuristic to assign points to subproblems

```
Require: Assume point-cluster assignments $\hat{y}_{ik}$ are given and $\mathcal{S} = \{1, \ldots, S\}$ is a given set of subproblems
1: Set $\mathcal{N}_s = \emptyset \ \forall s \in \mathcal{S}$; Set $s = 1$;
2: for each cluster $k \in \mathcal{K}$ do
3:     for each point $i : \hat{y}_{ik} = 1$ do
4:         $\mathcal{N}_s = \mathcal{N}_s \cup \{i\}$
5:         $s = s + 1$; if $s > S$, then $s = 1$;
6:     end for
7: end for
```

Algorithm 2 Randomly assign points to subproblems

```
Require: Assume $\mathcal{S} = \{1, \ldots, S\}$ is a given set of subproblems
Require: Assume the number $N_s$ of points per subproblem is the same for all subproblems $s$
1: Let $U_i$ be a uniform $(0, 1)$ random variable associated with point $i \in \mathcal{N}$
2: Let $\mathcal{N}^{\text{Sorted}}$ be the set of all points sorted in non-increasing or non-decreasing order according to $U_i$
3: Assign point $i \in \mathcal{N}^{\text{Sorted}}$ to subproblem $s = \lceil \texttt{ord}(i)/N_s \rceil$, where $\texttt{ord}(i)$ returns the ordinal position of
   $i \in \mathcal{N}^{\text{Sorted}}$
```

4.2.3 K-means computational experiments

We generated nine clustering instances with 100, 500, or 1000 points in 2, 3, or 10 dimensions for a total of 27 instances. We then attempted to solve each instance with $K \in \{3, 5, 10\}$ clusters. A warm-start solution was provided for each instance using the best solution found from calling the `kmeans` function in MATLAB with 100 replications. It is likely that the warm-start solution is optimal or near optimal. Our partition relaxations, called `TresPapas`, require a partition and time limit per subproblem. We assign 20 points to each subproblem in Algorithms 1 and 2, a number that was determined through minimal experimentation. Each subproblem is given a time limit of $\max\{60, \texttt{TOTAL\_TIME\_LIMIT}/|\mathcal{N}_s|\}$ seconds. All models and algorithms were coded in AIMMS version 4.13.1.204 and solved serially. Each method was given a time limit of $\texttt{TOTAL\_TIME\_LIMIT} = 900$ seconds (15 minutes).

Table 5 summarizes the solvers/methods that we compared. We compared three prominent commercial solvers - BARON 15, CPLEX 12.6.3, and Gurobi 6.5 - all of which can solve MIQCP formulations against our partition relaxation. Each commercial solver was tasked with solving the Big-M formulation (26), the Big-M formulation (26) with constraints (26c) implemented as SOS1, and the hull reformulation (27). All three of these models also include constraints (28) and (29). The same number of binary variables are present in both formulations (BM and HR); the main difference is the number of continuous variables and constraints. Note that BARON does not handle/accept SOS constraints, hence we could not solve a BM-SOS variant for BARON. Limited experimentation showed that the hull reformulation with SOS1 constraints was roughly the same. Finally, out of fairness to CPLEX, the following remark is in order:

Algorithm	**Description**
`BARON BM`	BARON 15 solving Big-M Model (26)
`CPX BM`	CPLEX 12.6.3 solving Big-M Model (26)
`GRB BM`	Gurobi 6.5 solving Big-M Model (26)
`CPX BM-SOS`	CPLEX 12.6.3 solving Big-M Model (26) with constraints (26c) implemented as SOS1
`GRB BM-SOS`	Gurobi 6.5 solving Big-M Model (26) with constraints (26c) implemented as SOS1
`BARON HR`	BARON 15 solving Hull Reformulation (27)
`CPX HR`	CPLEX 12.6.3 solving Hull Reformulation (27)
`GRB HR`	Gurobi 6.5 solving Hull Reformulation (27)
`TresPapas Det`	Call Algorithm 1 so that $\lvert\mathcal{N}_s\rvert = 20$; Solve each subproblem with `CPX BM`
`TresPapas R`	Call Algorithm 2 ten times so that $\lvert\mathcal{N}_s\rvert = 20$; Solve each subproblem with `CPX BM`; R-Mean/R-Min/R-Max = mean/min/max improvement over all 10 trials

Table 5: Algorithms compared.

Remark 5. The AIMMS-CPLEX 12.6.3 interface would **not** keep the warm-start solution in memory when Constraints (26c) were implemented with the property SOS1, hence this could unfairly handicap CPLEX as it was forced to spend time searching for a primal solution.

Figure 4 summarizes the instance-by-instance results in Table 6. In all instances, the MATLAB-provided warm-start solution was never improved upon and its objective function value was used as the upper bound in the optimality gap calculation. It is immediately clear from Figure 4 that the commercial solvers struggle to close the optimality gap for even the smallest instances. Meanwhile, our partition relaxation (deterministic or randomized) is significantly better than all commercial solvers at improving the lower bound. As more clusters are included (i.e., as K increases), the ability to close the gap decreases quickly. In these instances, the `TresPapas` approaches have difficulty making bound improvements, but at least make some progress in contrast with the commercial solvers which make virtually no improvements. Based on limited testing, we conjecture that the differences would be even more pronounced with a longer time limit.

When comparing the deterministic and randomized `TresPapas` approaches in Table 6, it is interesting to note that the deterministic variant outperforms the randomized one on average. Meanwhile, the randomized version appears to yield relatively stable results as can be seen by taking the difference of the R-Max and R-Min columns. This consistency is positive news since a randomized version is remarkably easy to implement. It would be interesting to see if the same consistency emerges in other problem classes.

We close this section by making some higher-level remarks. First, we believe that the K-means clustering problem is a prototypical MIQCP and, while one may not expect a generic MIQCP solver to be able to solve such problems as fast as, say, knapsack problems, it seems fair to expect such solvers to be able to make modest improvements on small instances. The instances considered here are quite modest compared to what is found in the machine learning literature. Second, it is surprising that our approach, which is quite general as it applies to almost all convex disjunctive programs and therefore to a large number of convex MINLPs, is able to make significant improvement relative to state-of-the-art solvers. Third, our approach is inherently parallelizable and thus we expect further gains from a parallel implementation.

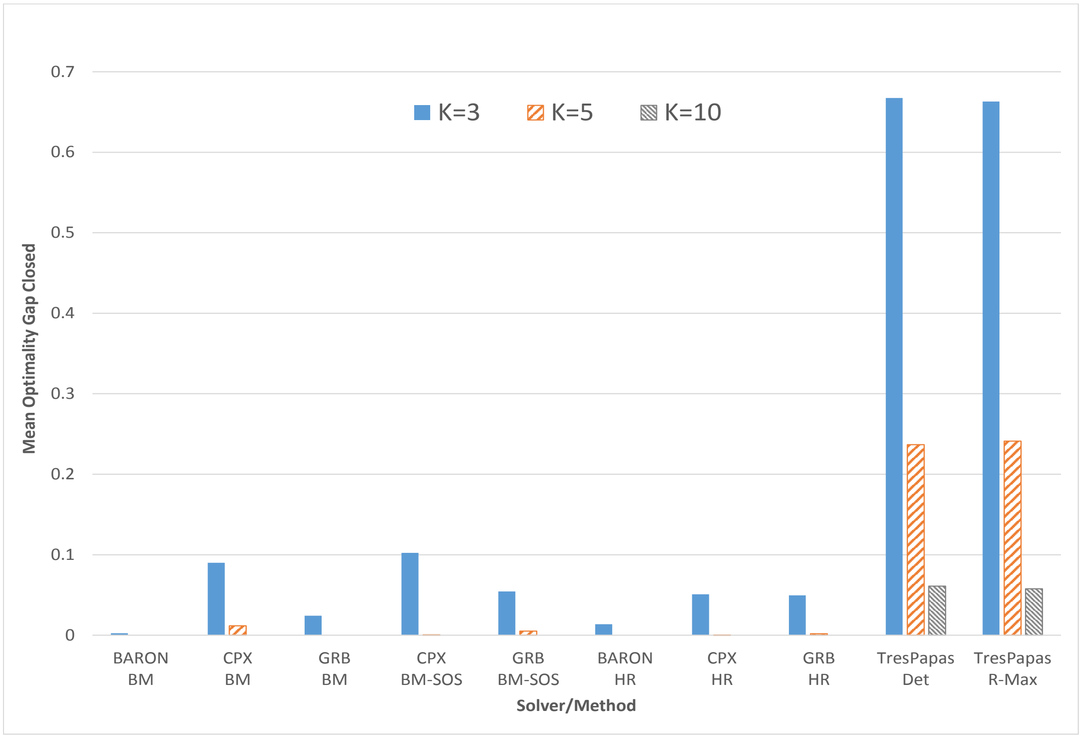

Figure 4: Comparison of average optimality gap closed for various solvers and methods on K-means instances.

5 Conclusions and future research directions

Basics steps are not a regularly used tool in the MIP repertoire. Nevertheless, they offer a very simple way to improve the formulation (i.e., the continuous relaxation) of a convex MINLP represented as a convex disjunctive program. In this work, we introduced the notion of a pseudo basic step to determine guaranteed bounds on the improvement from a basic step and showed that pseudo basic steps can make significant bound improvements relative to state-of-the-art commercial mixed-integer programming solvers on several sets of conic quadratic disjunctive programming instances. Given the growing excitement for solving convex MINLPs and the success of our approach, we hope to revive interest in basic steps and other techniques outside of the mainstream MIP toolbox.

Despite its efficacy to generate a tighter formulation, a basic step (as introduced by Balas [2]) requires one to intersect two disjunctions, thus forming a new single disjunction, possibly with more disjuncts, and then to re-solve the resulting hull relaxation. Practically speaking, a pseudo basic step retains some of the discrete features in the original disjunctive program (or ancestral disjunctive program in an iterative approach) by intersecting two or more disjunctions, and requires the solution (not necessarily to provable optimality) of a small convex disjunctive program. Today, some readers may argue that solving a convex MINLP, however small, is prohibitively expensive. We are more optimistic and believe that in the next two decades, our ability to solve convex MINLPs will witness roughly the same orders-of-magnitude improvement that computational MILP has experienced over the past two decades. In fact, we are hopeful that the adage

					BM			BM-SOS		HR			TresPapas			
Instance	N	D	K	z^{Best}	`BARON`	`CPX`	`GRB`	`CPX`	`GRB`	`BARON`	`CPX`	`GRB`	Det	R-Mean	R-Min	R-Max
1	100	2	3	3.14	0.01	0.35	0.12	0.40	0.30	0.06	0.25	0.24	**0.93**	0.79	0.70	0.85
2	100	2	5	1.91	0	0.05	0	0	0.05	0	0	0.02	0.61	0.58	0.50	**0.67**
3	100	2	10	0.75	0	0	0	0	0	0	0	0	0.04	0.03	0.02	**0.04**
4	100	3	3	4.68	0.01	0.35	0.09	0.35	0.19	0.07	0.19	0.21	**0.87**	0.83	0.81	0.87
5	100	3	5	3.14	0	0.05	0	0	0	0	0	0	0.55	0.49	0.41	**0.55**
6	100	3	10	1.79	0	0	0	0	0	0	0	0	**0.10**	0.08	0.07	0.10
7	100	10	3	12.68	0	0.08	0	0.1	0	0	0	0	0.64	0.61	0.56	**0.66**
8	100	10	5	9.70	0	0	0	0	0	0	0	0	0.05	0.05	0.04	**0.05**
9	100	10	10	7.86	0	0	0	0	0	0	0	0	**0.16**	0.14	0.14	0.16
10	500	2	3	15.02	0	0.02	0	0.01	0	0	0.02	0	0.75	0.72	0.67	**0.76**
11	500	2	5	8.62	0	0	0	0	0	0	0	0	**0.40**	0.33	0.29	0.39
12	500	2	10	4.00	0	0	0	0	0	0	0	0	**0.03**	0.02	0.02	0.02
13	500	3	3	17.70	0	0.01	0	0.04	0	0	0	0	0.81	0.81	0.79	**0.84**
14	500	3	5	13.44	0	0	0	0	0	0	0	0	**0.17**	0.14	0.13	0.17
15	500	3	10	8.76	0	0	0	0	0	0	0	0	**0.04**	0.04	0.03	0.04
16	500	10	3	67.10	0	0	0	0.01	0	0	0	0	**0.29**	0.26	0.25	0.28
17	500	10	5	44.40	0	0	0	0	0	0	0	0	0.03	0.03	0.03	**0.03**
18	500	10	10	39.51	0	0	0	0	0	0	0	0	**0.09**	0.08	0.07	0.08
19	1000	2	3	48.01	0	0	0	0	0	0	0	0	**0.76**	0.73	0.72	0.75
20	1000	2	5	25.05	0	0	0	0	0	0	0	0	0.19	0.17	0.15	**0.19**
21	1000	2	10	11.83	0	0	0	0	0	0	0	0	**0.02**	0.01	0.01	0.02
22	1000	3	3	29.21	0	0	0	0	0	0	0	0	0.81	0.79	0.77	**0.81**
23	1000	3	5	20.69	0	0	0	0	0	0	0	0	**0.11**	0.07	0.06	0.09
24	1000	3	10	14.29	0	0	0	0	0	0	0	0	**0.02**	0.02	0.01	0.02
25	1000	10	3	119.21	0	0	0	0	0	0	0	0	0.13	0.12	0.11	**0.14**
26	1000	10	5	87.56	0	0	0	0	0	0	0	0	0.02	0.01	0.01	**0.02**
27	1000	10	10	79.13	0	0	0	0	0	0	0	0	0.04	0.04	0.04	**0.04**
				Mean	0.00	0.03	0.01	0.03	0.02	0.00	0.02	0.02	**0.32**	0.30	0.27	0.32

Table 6: Fraction of optimality gap closed in 900 s for K-means clustering instances.

"Just MIP it!" [12] will soon apply to convex MINLPs as easily as it applies to MILPs today.

There are several fruitful pathways for additional research. Algorithmically, the question remains: How should basic steps be optimally chosen? Just as it is difficult to determine the best branching rule or cut in MIP, we conjecture that this question does not have a trivial answer. Nevertheless, there are likely some guiding principles that could help in computations. Another open question is: How can cuts generated from pseudo basic steps be included in a branch-and-cut framework? As far as applications are concerned, since our approach was successful when focused exclusively on MIQCP formulations, we hope to explore if there is even greater potential when applied to more general convex MINLP problems. Theoretically, it would be interesting to better understand the correspondence between the convex disjunctive program (14) associated with a pseudo basic step and the various cut generating optimization problems used, for example, to generate lift-and-project cuts. In addition, since much of the cutting-plane theory associated with linear disjunctive programming relies on two-term disjunctions, with some notable exceptions, e.g., Perregaard and Balas [16] consider lift-and-project cuts from multi-term linear disjunctions, it may be promising to investigate cut generating mechanisms for multi-term convex disjunctions given the success of our approach.

Acknowledgments

We wish to thank Ignacio Grossmann, Nick Sawaya, Myun-Seok Cheon, and Ahmet Keha for their feedback on a preliminary manuscript. We are grateful to two anonymous referees whose perceptive comments significantly improved the quality of the paper.

References

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] E. Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[3] E. Balas. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic Discrete Methods*, 6(3):466–486, 1985.

[4] E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1):3–44, 1998.

[5] E. Balas. *Disjunctive Programming*, pages 283–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[6] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming*, 58(1-3):295–324, 1993.

[7] P. Belotti, P. Bonami, M. Fischetti, A. Lodi, M. Monaci, A. Nogales-Gómez, and D. Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, pages 1–22, 2016.

[8] A. Ben-Tal and A. Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[9] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 2nd edition, 1999.

[10] P. Bonami, A. Lodi, A. Tramontani, and S. Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 151(1):191–223, 2015.

[11] S. Ceria and J. Soares. Convex programming for disjunctive convex optimization. *Mathematical Programming*, 86(3):595–614, 1999.

[12] M. Fischetti, A. Lodi, and D. Salvagnin. Just MIP it! In *Matheuristics*, pages 39–70. Springer, 2009.

[13] I. E. Grossmann and F. Trespalacios. Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming. *AIChE Journal*, 59(9):3276–3295, 2013.

[14] M. Guignard. Lagrangean relaxation. *Top*, 11(2):151–200, 2003.

[15] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. John Wiley & Sons, 1988.

[16] M. Perregaard and E. Balas. *Generating Cuts from Multiple-Term Disjunctions*, pages 348–360. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[17] R. Raman and I. E. Grossmann. Modelling and computational techniques for logic based integer programming. *Computers & Chemical Engineering*, 18(7):563–578, 1994.

[18] J. P. Ruiz and I. E. Grossmann. A hierarchy of relaxations for nonlinear convex generalized disjunctive programming. *European Journal of Operational Research*, 218(1):38–47, 2012.

[19] N. Sawaya. *Reformulations, relaxations and cutting planes for generalized disjunctive programming*. PhD thesis, 2006.

[20] N. Sawaya and I. Grossmann. A hierarchy of relaxations for linear generalized disjunctive programming. *European Journal of Operational Research*, 216(1):70–82, 2012.

[21] R. A. Stubbs and S. Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical programming*, 86(3):515–532, 1999.

[22] F. Trespalacios and I. E. Grossmann. Algorithmic approach for improved mixed-integer reformulations of convex generalized disjunctive programs. *INFORMS Journal on Computing*, 27(1):59–74, 2014.

[23] F. Trespalacios and I. E. Grossmann. Lagrangean relaxation of the hull-reformulation of linear generalized disjunctive programs and its use in disjunctive branch and bound. *European Journal of Operational Research*, 253(2):314 – 327, 2016.

[24] J. P. Vielma and G. L. Nemhauser. Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Mathematical Programming*, 128(1-2):49–72, 2011.